Massachusetts Institute of Technology
Lincoln Laboratory

A Study of Gaps in Network Knowledge Synthesis

George Baah
Hamed Okhravi
Shannon Roberts
Richard Skowyra
William Streilein

Technical Report 1195

October 18, 2015

This work is sponsored by the NSA's Cyber Defense Research & Technology Team under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government. Distribution A: Public Release

Lexington Massachusetts

This page intentionally left blank.

TABLE OF CONTENTS

This page intentionally left blank.

LIST OF FIGURES

This page intentionally left blank.

LIST OF TABLES

This page intentionally left blank.

1. EXECUTIVE SUMMARY

Network Knowledge Synthesis (NKS) refers to effective use of network defense information for cyber assessment and management. The vision of NKS is to achieve better informed situational awareness leading to superior cyber defense. Five major components are necessary to achieve this vision: sensor placement, data collection, data filtering, data analysis and sense making, and information sharing. By reviewing the state of the art for each of these components, we identify high-priority, short-term research objectives for NKS components, which include: collection of small, indicative, and symptomatic network data; connecting identities at multiple layers; ensuring the authenticity of collected data; identifying the ideal semantic layer for each type of data; developing scalable and decentralized filters; developing fast analysis algorithms that can operate in a malicious environment; testing such algorithms in real-world networks; and sharing properly anonymized network "knowledge" rather than raw data. These efforts will constitute the basic blocks of an effective NKS system.

This page intentionally left blank.

2. INTRODUCTION

Network knowledge synthesis (NKS) is concerned with the effective use of network defense information for cyber assessments and management. The ultimate goal of NKS is to achieve better informed situational awareness leading to superior cyber defense. Research in this area focuses on the application of external environmental information to internal network assessment information with the goal of improving risk management decisions and developing enhanced shared situational awareness capabilities.

2.1 GOALS

This report identifies the major gaps and challenges that currently exist in NKS. By analyzing the state of the art in NKS research, we identify and prioritize the important gaps that need to be researched in order to construct a system that effectively uses network defense information for cyber assessments and management.

The goal of this study is to find technical gaps within research projects that can be addressed in the short term to make effective headway in building a NKS system.

2.2 SCOPE

For this report, we focus our study on prominent cybersecurity research. To this end, we evaluate the publications in top-tier security conferences, including IEEE Symposium on Security and Privacy (Oakland), ACM Computer and Communications Security (CCS), Network and Distributed Systems Security (NDSS), USENIX Security, Symposium on Research in Attacks, Intrusions, and Defenses (RAID), and Annual Computer Security Applications Conference (ACSAC). We have incorporated papers from other relevant conferences (including programming languages and operating systems conferences) as necessary.

Since cybersecurity research progresses very rapidly, we focus our study on the past four years, but include older publications if they constitute the state of the art in that area.

We identify the gaps in different components of NKS in Section 3, while excluding the areas covered in our previous "Attack Analysis" [27], "Web Security Countermeasures" [5], and "Cyber Defense Automation" [4] reports.

Research areas that are explicitly in-scope for this report include, but are not limited to: situation awareness of assets and risk, data collection, sensors for collecting relevant data at the right abstraction layers, correlation of data for improved detection, data analysis and sense making, and information sharing. In this report, we focus on these activities as they relate to an enterprise network.

Many of the techniques discussed in this report address NKS gaps on traditional enterprise networks by utilizing Software-Defined Networking (SDN), which many existing networks can be augmented to use. In SDN, packets are forwarded through the network by matching their head-

ers against a set of pattern-match rules and associated actions. A network controller (i.e., SDN controller) on a physically separated control plane adds and removes these rules to network devices in response to sensed events. For example, if a packet arrives whose header does not match any existing rule, it can be sent to the controller for analysis. In response, the controller emits rules to one or more switches specifying how to handle packets of that type in the future. As a result, SDNs separate the control plane (i.e., SDN controller) from the data plane (i.e., switches). The best-supported SDN technology is OpenFlow [15], which is now available from most COTS networking equipment vendors (e.g., Juniper Networks, Broadcom, etc.). In many cases, existing non-SDN switches can be converted to SDN via software patching. Note that although some of the gaps are discussed in the context of SDN because the state of the art technique for that component incorporates SDN, the gaps are generic, and they are equally applicable to non-SDN NKS systems.

Due to limited resources, low priority, desired sponsor focus, and treatment in previous reports, we exclude these potential NKS areas from this report:

- Attack detection: activities primarily focused on detection of malicious activities (covered in the "Attack Analysis" report [27])
- Social network research: analysis or measurements related to social network websites, e.g., Facebook, Twitter, etc. (low priority)
- Mobile ad-hoc networks: research focused on mobile ad-hoc networks (MANETs) and sensor networks (low priority)
- Internet-scale measurements: analysis and statistical surveys of large parts of the Internet; this area of research can be viewed as situation awareness of the Internet, rather than an enterprise network (not the desired focused and limited resources)

The rest of this report is organized as follows: Section 3 provides an overview of NKS and its components; Section 4 describes our methodology, gap selection, and gap prioritization processes; Section 5 describes the state of the art for each of the NKS components and their high-level gaps; Section 6 describes the overall summary and prioritization of the gaps; and Section 7 concludes the report.

3. OVERVIEW OF NETWORK KNOWLEDGE SYNTHESIS

Proper network knowledge synthesis (NKS) is a system comprised of multiple components to enable better situation awareness. Each component supports and contributes to network knowledge synthesis. An NKS system begins with Sensor Placement, followed by Data Collection, Data Filtering, Data Analysis, and finally, Information Sharing, as shown in Figure 1.

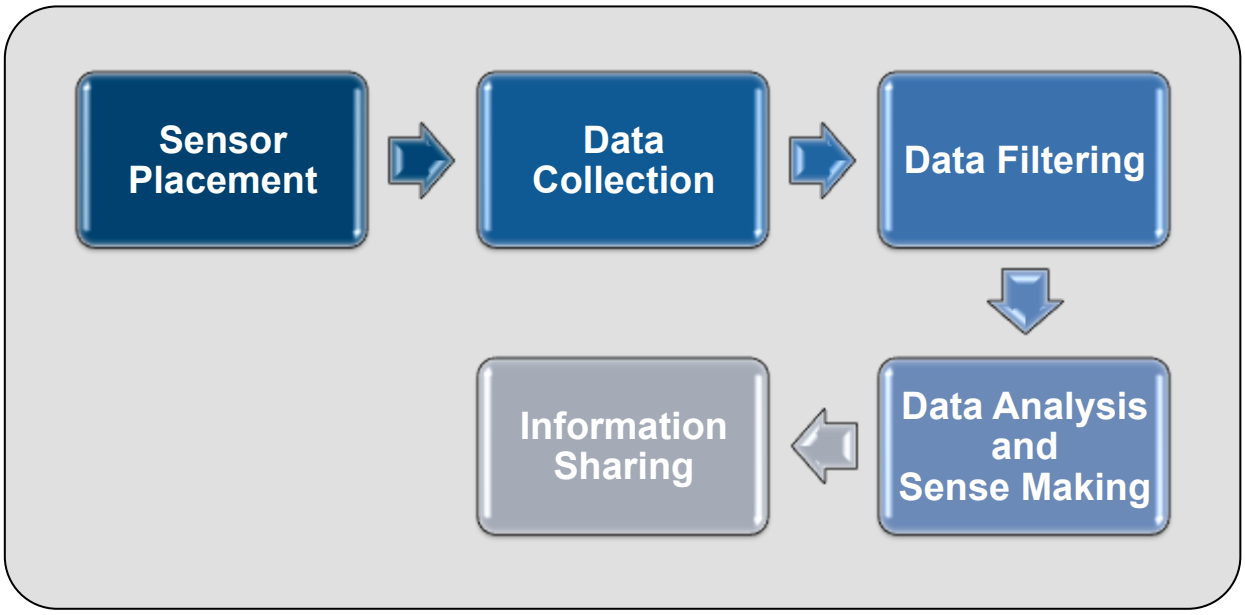

Figure 1. Components of a Network Knowledge Synthesis System

Sensor Placement is concerned with where sensors are placed on the network such that the correct type of data can be collected. Sensors refer to any hardware device or software code that is responsible for collecting raw data in the network. A network tap, an end-host packet counter, and an authentication log collection agent are all examples of sensors. Within this component, it is important to consider how much data the sensors will collect, where the sensor is located with respect to the data storage facility, where data analysis occurs, and how the sensor will perturb network performance. For example, sensors that collect packet headers can be placed on switches, routers, software-defined networking (SDN) controllers, and/or end hosts. Each of these possible choices has its implications regarding scalability, accuracy, overhead, and difficulty to evade.

The Data Collection component concerns the type and quality of data that is collected for network knowledge synthesis. Volume, semantic layer, and authenticity of collected data are important considerations. Other important issues related to data collection include transfer, storage, encoding, compression, and processing.

Data Filtering is designed to reduce and/or refine the raw dataset so that it is more tractable and useful for analysis. Though this is a frequently overlooked step, it can have an important effect on the quality and quantity of knowledge that can be extracted. For example, if operators are forced to sift through many false positives before identifying a true detection, the tool or sensor will be abandoned.

Data Analysis is the largest component within NKS as it deals with making sense of the data. Though there may only be one dataset, the types of data analysis that can be performed on the dataset are extensive. Similarly, the number of conclusions that are drawn from data analysis is voluminous. As such, it is important to ensure that the type of analysis performed achieves the goal of improving situation awareness for a given threat model.

Lastly, Information Sharing focuses on dissemination of information about network status, events, alerts, and intrusions to other organizations. Information sharing can be imperative in ensuring that multiple organizations do not succumb to the same attacks. At the same time, though the concept of information sharing is not novel, it is not widely practiced as there are many challenges with respect to effective deployment.

4. METHODOLOGY

We have conducted a systematic review of the state of the art in the development of each of the five NKS areas, covering promising research that has been done within the last four years. In the course of our review, we have identified a number of existing gaps that remain towards achieving network knowledge synthesis that allows for network assessment and improved risk management.

4.1 GAP DISCOVERY PROCESS

For the purposes of this study, we focus on the open literature and publicly known NKS techniques. We reviewed papers and studies, mainly from the prominent security conferences such as the IEEE Symposium on Security and Privacy (Oakland), Network and Distributed Systems Security (NDSS), ACM Computers and Communications Security (CCS), USENIX Security Symposium, Annual Computer Security Applications Conference (ACSAC), and the Symposium on Research in Attacks, Intrusions, and Defenses (RAID).

We initially surveyed the titles and abstracts for the past four years of each conference's proceedings. We then down-selected this large list of papers based on their relevance to NKS. We read the papers from this refined list and extracted any relevant gaps based upon our own experience, the context of the paper, and the stated research contribution.

4.2 GAP SELECTION PROCESS

The gap selection process was performed during team meetings and began with individual team members independently determining if a paper was worthy of carrying forward. After gaps from each paper were identified, a majority vote was then used to determine whether to keep each gap.

Note that there is a significant non-uniformity in the amount of research being done on NKS components. Some components of NKS (e.g., sensor placement) are not widely studied in the academic community and have very few relevant papers. In these cases, the gaps which have been identified may not have corresponding academic work. This is an indication that no current work exists that tries to address the gap; it does not indicate that the gap itself is minor or does not exist. Moreover, we have included *all* papers from the sources listed above that relate to NKS components, so the scarcity of papers related to a specific component (e.g., sensor placement) is a good indication of the lack of research attention to that component.

4.3 GAP TREATMENT AND CLASSIFICATION PROCESS

There are three major types of gaps in existing network knowledge synthesis systems:

- Technology Gap: an effective system does not yet exist
- Deployment Gap: systems have been proposed or implemented (open source or commercial), but all of the existing systems have impracticalities that impede their adoption

- Practice Gap: practical and effective systems exist in the community, yet they are not adopted widely, perhaps because of the lack of awareness or incentives

In this report, we primarily focus on deployment and technology gaps. Note, however, that the distinction between these areas is not well-established and there are many gray areas. For example, the amount of acceptable overhead varies from system to system, which means that a deployment gap in one system can be interpreted as a practice gap in another. Moreover, each deployment gap eventually points to a technology gap. For example, knowing that all existing systems have high overhead (a deployment gap) points to the fact that faster systems have to be developed (a technology gap).

4.4 GAP PRIORITIZATION PROCESS

To prioritize the gaps within each component, we used a subjective ranking system with two components: likelihood and impact. Likelihood is the frequency at which this gap occurs in current NKS systems. For example, some analysis tools utilize active techniques, but as this does not occur often, it has a low likelihood. Impact is how severely the gap impedes NKS capabilities. For example, the nonreduction of storage size due to filtering is not a major hindrance to NKS as one can perform the analysis in real time if the other components of NKS are in place; as such, it has low impact. As this was a subjective ranking, both likelihood and impact were given qualitative values of low, medium, and high.

To generate this ranking, first, the team identified the highest priority gaps for each component within NKS based on the number of papers highlighting the gap, whether the gap has truly been solved, and whether current technology exists that can solve the problem. Next, individual team members subjectively ranked each gap according to its likelihood and impact. The rankings for each gap were consolidated to produce a final ranking for each gap, i.e., when all team members agreed, the final ranking was recorded, but when team members disagreed, a discussion ensued until all team members agreed on a consensus and then the final ranking was recorded. Note that our priority ranking is a subjective computation based on the qualitative ranking of our team members. Other researchers are likely to compute different priorities based on their scoring of the gaps.

5. GAPS AND RESEARCH DIRECTIONS

5.1 GAPS IN SENSOR PLACEMENT

5.1.1 High-Level Gaps

1. Strong trade-off between how much data is collected and overhead
2. Sensors require binding identities across multiple layers
3. Sensors are spatially/temporally static
4. Many network devices are black boxes
5. Sensors require collection across multiple layers

5.1.2 Description of Gaps

1. Strong trade-off between how much data is collected and overhead.

   Sensors are traditionally placed on the network edge (e.g., at firewalls, gateways, or web proxies) where traffic volume is highest, rather than distributed across the network. This forces a tradeoff between lightweight sensing of all packets at line rate and collecting more data per-packet versus being forced to sample from traffic and possibly missing important events.

2. Sensors require binding identities across multiple layers.

   Sensors collect network identifiers like IP and MAC addresses from packet headers, while application logs collect user names and other high-level identifiers. Without a secure binding between these identifiers, mapping sensed network events to user or application actions is unreliable and vulnerable to spoofing.

3. Sensors are spatially/temporally static.

   Sensors, once placed, are generally fixed in time, space, or both. This makes their behavior predictable to an attacker, who can then operate outside of the sensor's range (space) or its sampling period (time). This gap is present in the Symbiote technique [10].

4. Many network devices are black boxes.

   Network routers, switches, and middleboxes usually run proprietary, closed-source firmware whose features and capabilities are only partly documented. This makes it difficult to insert new sensors into these devices, and makes the limitations of any provided sensors difficult to understand. This gap is present in the Symbiote technique [10].

5. Sensors require collection across multiple layers.

   Effective collection of NKS data should also happen across multiple software stack layers including the network, hardware, operating system, and applications. For example, to answer the typical query of "who connected to this service?", it is not enough just to collect network packets. One, in addition, has to collect server logs, login names from the operating system, and application logs. Current NKS sensors are often limited to network collection only.

There is little state of the art work on sensor-placement in the academic community. Below, we cover the only relevant technique to appear in top-tier conferences. It is addresses gap 1 but suffers from gaps 3 and 4.

5.1.3 Technique: Symbiotes

Description of Technique
Embedded devices such as routers, switches, and firewalls form a communication substrate for the Internet. These devices connect servers and hosts over the Internet. However, current defensive measures only protect hosts or servers while leaving the communication substrate prone to exploitation.

The authors present a technique that automatically hardens switches, routers, and firewalls by injecting a software construct known as a Symbiote [10] into the unused address space portion of the host program executing in legacy embedded devices. The Symbiote effectively turns embedded devices into sensors that detect unauthorized modifications of static areas within the host program's address space. The Symbiote has five fundamental security properties:

1. has full visibility into the code and execution state of the host program
2. can passively monitor or actively react to events observed at runtime
3. executes alongside the host program
4. is an autonomous entity, which has been hardened to defend against removal or unauthorized modification once it is injected into the address space of the host program
5. is differentiable because Symbiotes are randomized and mutated each time they are created, thereby ensuring that no two instantiations of the same Symbiote are the same; randomizing and mutating the Symbiote renders attacks that rely on predictability of code structures in the Symbiote ineffective.

To demonstrate the efficacy of their approach, the authors use Cisco firmware and hardware as the platform by injecting a Symbiote into a Cisco router. In addition to the Symbiote-protected router, the authors also build a monitoring station and a capture and analysis system that automatically collects and analyzes forensics data once an alarm is triggered by the Symbiote. The authors

show that given the right Symbiote scheduler (the scheduler impacts the resource consumption of the Symbiote), the impact of Symbiotes on the control plane can be significantly reduced without adversely affecting the detection latency of the Symbiotes. The authors show that the Symbiote is able to detect unauthorized modifications of memory in 450 ms.

Technique-Specific Gaps

Although the Symbiote approach is interesting, it has its gaps. The Symbiotes cannot detect attacks that do not modify static memory regions in the host program's address space. Because of this, an attacker can disable a Symbiote before actually executing their payload. The authors used a Cisco router as an example, but it still remains to be seen how effective their technique is on other embedded devices. The efficiency and effectiveness of the Symbiote is based on the Symbiote scheduler and determining a priori the right scheduler to use is still unknown. Also, it is difficult to determine how scalable, i.e., in terms of CPU usage and detection latencies, the technique is in an enterprise network with many routers and switches.

5.2 GAPS IN DATA COLLECTION

5.2.1 High-Level Gaps

1. Collection at line speed is very hard
2. Assumes a benign environment
3. Collection occurs at the wrong semantic level
4. Analysis does not happen at collection location, resulting in transfer overhead
5. Unscalable to store collected data
6. Efficient techniques depend on type of data collected
7. Data must be collected to find interesting events, but one must know what data to collect to find events —a Catch-22
8. In-band collection perturbs the network
9. Requires manual effort to specify what data to collect
10. Cannot work on encoded/compressed data

5.2.2 Description of Gaps

1. Collection at line speed is very hard.

   Routers, switches, and middleboxes must parse packet headers in order to route traffic, which incurs end-to-end latency costs. These costs (or costs for dedicated hardware to mitigate the gap) are substantially increased if the device must also duplicate the header or even the entire packet, which is often an order of magnitude larger than the header. This gap is present in Netsight [17], OFRewind [33], and Peek-a-Boo [14].

2. Assumes a benign environment.

   Data collection systems assume that all data is a genuine reflection of network conditions. In many cases, however, an attacker can falsify or corrupt data to reflect a poisoned view of network state (e.g., ARP posioning or DHCP spoofing). Additionally, the collection system itself might be targeted by exploits if it parses any of the collected data. This gap is present in NetAssay [12], Netsight [17], OFRewind [33], and Peek-a-Boo [14].

3. Collection occurs at the wrong semantic level.

   Network data collection generally is focused on packets headers and payloads, but analysis focuses on application-layer events like logins, database accesses, etc. Mapping the low-level

data (packets sent from A to B) to high-level events (a user has logged in) can be very difficult. This gap is present in Netsight [17], OFRewind [33], SE-Floodlight [29], and Peek-a-Boo [14].

4. Analysis does not happen at collection location, resulting in transfer overhead.

   Data collection occurs either in-network on switches, routers, and taps, or on end-hosts. Moving this data to a central storage and analysis server uses the same network that is being monitored, which makes scalability very difficult. For example, if all network traffic is captured then network bandwidth usage will double. This gap is present in NetAssay [12], Netsight [17], OFRewind [33], and NetStore [16].

5. Unscalable to store collected data.

   Modern networks have bandwidths measured in gigabits or tens of gigabits per second. Thus, storing even one day of network traffic can easily require tens of terabytes of storage. This gap is present in OFRewind [33] and NetStore [16].

6. Efficient techniques depend on type of data collected.

   Data can be collected efficiently if its structure is known and uninteresting portions can be identified and ignored. This varies by the type of data being, e.g., IP Addresses may only have certain subnets of interest, TCP ports may only be worth collecting if the port is not ephemeral, etc. If the kind of data to be collected changes over time, substantial control overhead may be required to change which efficient data collection schemes are in place. This gap is present in SE-Floodlight [29], Observing Unobservable Network Communications [18], Peek-a-Boo [14], and NetStore [16].

7. Data must be collected to find interesting events, but one must know what data to collect to find events —a Catch-22.

   Since it is often impossible or unscalable to collect all network data, operators can collect only an 'interesting' subset that contains relevant events. However, this requires *a priori* knowledge of future events, which is only available for predictable events and not attacks or anomalies. This gap is present in NetAssay [12], Netsight [17], OFRewind [33], and NetStore [16].

8. In-band collection perturbs the network.

   Collecting network data often uses the same computation and storage resources that are used for packet forwarding. This can lead to increased latencies and lowered bandwidth, especially if collected data is also transferred over the network being monitored. Moreover, this collection changes the properties of the network (e.g., bandwidth usage, open ports, valid data streams, etc.) being monitored. This gap is present in Netsight [17] and OFRewind [33].

9. Requires manual effort to specify what data to collect.

In order to determine if a packet is worth logging, data collection systems require operators to define selection criteria. These are often in the form of low-level regular expressions or wildcarded statements that are difficult to write correctly and even more difficult to debug or verify. This gap is present in NetAssay [12], Netsight [17], and OFRewind [33].

10. Cannot work on encoded/compressed data.

    To remain scalable, data collection systems cannot collect all network traffic. Instead, they must rely on a set of selection criteria to identify packets of interest. If the data in these packets is compressed, encrypted, or encoded, however, the system cannot recognize features of interest and will discard the packet rather than collecting it. This gap is present in NetAssay [12], OFRewind [33], and NetStore [16].

The following techniques represent cutting-edge academic research on data collection for network knowledge synthesis. While all of the gaps described above are present in at least one technique, gaps 2, 3, 4, 6, and 7 are present in every technique that develops a general collection framework for network data. Furthermore, gaps 10, 1, 2 ,7, and 8 are not addressed by any technique.

5.2.3 Technique: NetAssay

Description of Technique
Network operators monitor network traffic using low-level features, e.g., IP addresses. These low-level features do not allow operators to monitor based on intent, e.g., continuously monitor a user known to show suspicious web patterns. Intentional network monitoring allows operators to monitor traffic based upon higher-level principles such as the user, application, or device that is initiating the traffic or where the traffic is going [12].

NetAssay is a mechanism that allows for intentional network monitoring. It does so by allowing for dynamism and heterogeneity. With dynamism, as users, applications, and services do not map to static entities, a binding mechanism must be used to dynamically map higher-level abstractions to flow-space in realtime. Heterogeneity is required as the mapping between higher-level abstractions and low-level flow-space entities depend on many data sources, e.g., DNS and BGP [12].

NetAssay has two components: a control application and metadata engines. The control application is written in a Pyretic-style language to augment Pyretic with NetAssay capabilities. The metadata engines translate a NetAssay policy into a Pyretic policy and dynamically update the policies. The metadata engines will depend on many different mappings, e.g., a high level abstraction to match specific users may require low-level entities that track multiple source IPs. The exact low-level entities correspond to flow table entries for network switches. The control application essentially then takes the rules created by the metadata engines and pushes them to the network switches [12].

Technique-Specific Gaps
NetAssay is an ongoing project; it is not yet a fully deployable tool. More specifically, in its current form, NetAssay does not allow for operators to submit custom high-level abstractions. In addition, NetAssay does not guarantee to be accurate with respect to only delivering information of interest, e.g., an operator's request to monitor one user's streaming flows may result in extraneous flows. Lastly, NetAssay is not completely scalable for an actual network. It is in need of clever filtering mechanisms to ensure that policies are not recomputed each time a metadata engine is updated. There are also scalability concerns as any given rule may yield tens or more of flow table entries. Until these major issues are remediated, NetAssay cannot be tested for effectiveness, validity, etc. [12].

5.2.4 Technqiue: Netsight

Description of Technique
Netsight [17] is an SDN control-plane platform for debugging and diagnosing network conditions. It is based on recording and analyzing packet histories, which are per-packet logs of what switches were traversed, what header fields were modified, and what actions were taken on a packet at each switch, for all packets traversing the network. A Packet History Filter language (similar to regular expressions) allows packet histories of interest to be easily retrieved. Netsight requires OpenFlow or similar SDN architectures that support packet duplication, truncation, and tagging at switches. Duplicate packets are forwarded to Netsight logging servers for storage and processing. In order to minimize storage costs, Netsight uses aggressive differential (delta-based) compression to keep storage overheads at 10 —30 bytes per packet and CPU overhead at 0.1μs—20μs per packet. Bandwidth overhead from forwarding duplicate truncated packets is approximately 30%.

Using the Netsight platform, the authors implemented four network situational awareness tools:

- ndb —A network debugger that can define 'breakpoints' as packet history filters (PHF) matching erroneous behavior. Whenever these events occur, histories containing a trace of the error are returned. Errors include reachability, race conditions in flow rule installation, and incorrect packet modification.
- netwatch —A live invariant monitor that triggers alarms and provides traces whenever a specified invariant is violated. Supported invariants include isolation, absence of forwarding loops, waypoint routing, and max-path-length.
- netshark —A Wireshark-like path packet logger that enables filters to be set over entire packet histories (i.e., the progress of a packet over multiple hops).
- nprof —A network profiler that identifies, for a specific link, all sources of congestion (and their bandwidth) contributing to that link's utilization. These can be grouped by end-host/switch, or by packet history and header.

Technique-Specific Gaps
Netsight's 30% bandwidth overhead is substantial for production enterprise and datacenter networks. While the authors propose two methods for mitigating this overhead, they require either hardware modifications to switches (to perform compression at the collection point) or installation of software on end-hosts (to distribute packet history storage). In addition to dataplane overhead, Netsight's impact on controller latency (e.g., when responding to `Packet-In` event) is not evaluated. Some overhead must be present, however, as `flow-mod events` are intercepted and modified by the platform. Finally, Netsight's dataplane flow rule injector and forwarding state monitor must be implemented on a per-controller basis, and must have permission to modify all flow rules generated by the controller. This assigns substantial security privileges to Netsight and requires all controller applications to forward their flow rule modification through Netsight, rather than directly to the dataplane. Depending on how the controller and applications are implemented, this may necessitate substantial implementation changes to existing code.

5.2.5 Technique: OFRewind

Description of Technique
OFRewind [33] is an SDN-based data collection system, implemented using OpenFlow, to record and replay packets of interest on a network. Both data and control plane traffic is instrumented using this system, allowing OpenFlow messages as well as dataplane control traffic (e.g., DHCP) to be captured and replayed. OFRewind does not record all packet headers for scalability reasons, but instead allows administrators to define match rules for packets that should be captured (such as low-rate control data). OFRewind imposes minimal overhead on the rate at which the SDN controller can send flows. Due to the rate at which OFRewind must itself send flow updates to switches, however, flow rules may be dropped at switches due to queue overflow beyond the 200 rules/second boundary.

OFRewind has four modules that are added to an SDN. OFRecord acts as a proxy between OpenFlow switches and the controller, logging OpenFlow protocol and switch state messages (e.g., `Packet-In` and `Flow-Mod`). OFReplay can replay either messages from switches to the controller or vice versa, enabling either element to be emulated during testing or debugging. Datarecord records data-plane control and data traffic that is specified by an administrator as important or interesting. It is stored on dataplane-hosted storage servers. The recording mechanism relies on packet duplication via flow rule actions, and thus imposes network overhead. No evaluation of this overhead is presented. Datareplay re-injects recorded dataplane traffic in synchronization with OFReplay to re-inject the flow rules governing the replayed traffic.

Technique-Specific Gaps
OFReplay has a limited view of the network, which is defined ahead of time by an administrator. This makes it difficult to utilize the tool when the cause of an anomalous event is not already known or suspected. Furthermore, OFReplay's scalability with respect to imposed network overhead is not evaluated, but is likely to be substantial if significant dataplane traffic is duplicated without compression. Control-plane latency may also suffer, as OFReplay acts as a "shim" between dat-

aplane switches and the controller. Capturing and processing control plane messages could have significant latency and bandwidth overhead.

5.2.6 Technique: SE-Floodlight

Description of Technique
SE-Floodlight [29] is an SDN Controller built on Floodlight, a popular Java-based SDN controller framework. It provides a secure Northbound API and a Security Enforcement Kernel that provides authentication and access control of controller apps. Its privilege levels that a controller app can have are APP, SEC, and ADMIN (fewest to most privileges). Controller Apps and Northbound API users digitally sign flow-rule or sensor-access requests. If the signature is valid, the Security Enforcement Kernel grants or denies based on the app's role.

In order to detect and resolve situations where a rule conflicts with the security policy, SE-Floodlight uses an efficient formal abstraction called rule-chain conflict analysis. It detects and denies both direct conflicts between APP and SEC flow rules (DROP vs. FORWARD) and indirect conflicts in which a series of rules that use header rewriting combine to form a conflict. Conflicts are always resolved to favor the most privileged app involved, if the roles are unequal.

SE-Floodlight also maintains a secure network-audit log that captures any network changes, controller configuration changes, changes to the audit service, sensor data requests and replies, and authentication events. All entries are timestamped and include the security credential of the causal process. Access to the log is only available to applications with SEC or higher roles. SE-FLoodlight can be configured to halt if the audit log becomes too full or to drop logging of selected events.

Technique-Specific Gaps
SE-Floodlight is only implemented for one version of Floodlight. Other controllers, or future updated versions of Floodlight, may require manual updates or re-implementation. Additionally, SE-Floodlight has its own API, which is not composable with off-the-shelf controller modules. These would have to be modified to run on a network protected by the tool.

Scalability may become a problem in heavily dynamic networks. Since rule insertions are cryptographically authenticated, a high rate of network reconfiguration could impose substantial computational overhead on the controller.

Finally, SE-Floodlight provides no protection against attacks from the dataplane. In addition to exploits arising from parsing malformed packets, dataplane attacks can also poison controller datastructures with corrupted information about network state and configuration.

5.2.7 Technique: Observing Unobservable Network Communications

Description of Technique
This work [18] demonstrates low-cost, efficient methods to observe traffic hidden with "parrot" tools that disguise Tor traffic as benign traffic (Skype, VoIP, and HTTP). The specific tools considered are SkypeMorph (Skype), StegoTorus-Embed (Skype), StegoTorus-HTTP (HTTP), and CensorSpoofer (VoIP). The paper defines three classes of attack: passive (observation), active (man-in-the-middle)

and proactive (imitation of users by the network operator). Additionally, there are three classes of adversary considered: Local (control of a small network or single router), State-Level Oblivious (large-scale network access, but limited computational/storage resources), and State-Level Omniscient (large-scale deep packet inspection, unlimited storage/compute resources). The paper argues that local detectors are both easier to deploy and more effective against detection of imitators than global detectors working over aggregated data.

Skype-based parrots can be detected using several classes of attack. For passive analysis, TCP control side channel behavior is not emulated correctly by any known imitation tool. Skype-specific packet header and application-layer control traffic (e.g., login/logoff) can also be used to distinguish imitators. Active disruption by dropping UDP packets, delaying TCP packets, closing TCP channels, and blocking TCP ports will cause Skype imitators to react differently (or not at all) than actual Skype error-handlers.

HTTP-based parrots can be detected by observing correlated traffic patterns between a host and different web servers. In real HTTP traffic, correlation is low between different servers. In parrots, high correlation can be observed due to splitting of a single Tor connection over several HTTP connections. Correlation can be passively sensed over time, or actively measured by dropping packets. Parrot HTTP servers can also be identified proactively by sending malformed HTTP requests to the suspected server. No parrot correctly handles error conditions properly.

Finally, VoIP-based parrots can be actively identified by sending SIP invitation probes, invalid or malformed SIP packets, and dropping RTP packets. All of these interactions are handled differently by imitators than by a genuine SIP client.

Technique-Specific Gaps
Many of the attacks presented in this paper involve active manipulation of network flows in real time. This can be difficult or resource-intensive in large enterprise networks, especially when the manipulation must be performed at or near line rate. Additionally, many other attacks are proactive and involve probing/scanning of external servers suspected of supporting parrot systems. This may prompt legislative or policy issues depending on the legal jurisdiction of the respective networks.

5.2.8 Technique: Peek-a-Boo, I Still See you: Why Efficient Traffic Analysis Countermeasures Fail

Description of Technique
In this paper [14], seven traffic analysis techniques for website fingerprinting are studied with traffic analysis countermeasures in place. Two of them can distinguish which one of 128 websites in a closed-world environment is visited, with 80% accuracy. When the world is reduced to two websites, accuracy rises to over 98%. This result holds for all nine countermeasures that were investigated. The first technique, VNG++, uses a naive Bayes classifier over total session time, total bytes, and bytes in traffic bursts. The second uses a Support Vector Machine classifier over packet lengths, ordering of packets, and total bytes transferred.

The countermeasures that are studied fall into two classes. Packet-length padding increases packet lengths using either a constant amount of padding or an amount determined by the initial packet size (e.g., linear or exponential padding). Distribution-based padding either increases or decreases (via fragmentation) the length of a packet based on a target distribution associated with a different website than that visited.

The authors' results show that coarse-grained features (total time, total bandwidth, size of bursts) are sufficient to classify websites. Fine-grained information like packet sizes and inter-arrival times are not needed, thus their obfuscation hides nothing useful from the classifier.

Technique-Specific Gaps
The paper's results are not very surprising. All of the countermeasures that the authors studied rely on changing packet length, while the strongest classifiers do not even use packet length as a feature. Additionally, closed-world analysis of 128 websites say little about real web traffic in which a large number of sites are visited according to complex distributions. Similarly, the experiments were carried out in a laboratory setting. Real-world conditions and noise associated with actual Internet web browsing was not considered, nor was the vulnerability of these classifiers to such noise.

5.2.9 Technique: NetStore

Description of Technique
The increase in the sophistication of network attacks requires that security monitoring systems be able to store and query large amounts of network data quickly for effective network forensics. Currently, traditional row-oriented Relational Database Management Systems (RDBMS) are used to store data, but these systems have slow insertion and query rates. This makes RDBMS unsuitable for storing and querying large amounts of data for network forensics. For example, when querying an RDBMS, all the columns of a table have to be loaded into memory regardless of whether the columns are relevant to the queries. This introduces an I/O bottleneck that causes RDBMS to have slow query times.

The authors overcome this problem by proposing NetStore [16], a column-oriented database system. The intuition behind NetStore is that network-flow data exhibit certain characteristics. NetStore takes advantage of such characteristics to provide fast insertion and query rates of large amounts of data. For example, the authors observed that forensic queries access specific predictable attributes that are collected over longer periods of time. Given these characteristics, the authors identify five types of queries: (1) spot queries, (2) range queries, (3) aggregation queries, (4) spot aggregation queries, and (5) range aggregation queries. The authors optimize NetStore for the efficient evaluation of these queries.

In building NetStore, the authors store network-flow data by columns, where each column holds data for a single attribute of a network flow. To facilitate better compression and faster access, NetStore sorts the first column. Column arrangement is based on how likely a column will be accessed based on past data access query patterns. Note that columns can be accessed

independently of each other. Columns are further partitioned into segments to allow for greater flexibility for compression strategies and data accesses.

The authors compare NetStore to PostgreSQL and a column store database LucidDB. The authors found that NetStore outperforms PostgreSQL significantly, i.e., the authors found that NetStore is 10 times more efficient than PostgreSQL in terms of query speed. NetStore also performs better than LucidDB when storing less than six times the data. However, LucidDB performs better for repeated queries because it efficiently uses data stored in memory.

Technique-Specific Gaps

Although NetStore is more effective than PostgreSQL, it is not necessarily better than other RDBMS because other RDBMS implementations may use different algorithms in their storage and retrieval of network records. NetStore is also inflexible because it supports only a specific set of queries, and a limited subset of SQL syntax, potentially making some analyses of the stored network data infeasible. In addition, it is still not known whether NetStore can handle data that is larger than what was used in the experiments. Because NetStore is optimized for network flows with certain characteristics, it is not known how effective NetStore will be in the presence of other types of network data. It is also not clear whether NetStore can analyze its stored data in compressed format after reading it from disk.

5.3 GAPS IN DATA FILTERING

5.3.1 High-Level Gaps

1. Trade-off between resources and quality of data filtering
2. Filtering relies on thresholds from ground-truth, which may be unknown
3. Advanced filtering techniques require customized infrastructure
4. Efficient filters are shallow, while interesting features require deep inspection
5. Filtering does not necessarily reduce storage size

5.3.2 Description of Gaps

1. Trade-off between resources and quality of data filtering.

   Filtering separates relevant data from noise, and the results may contain false positives and false negatives. The false positive and false negative rates depend on how detailed the classifiers for relevant data are (e.g., hash functions or pattern matches). As false positive and false negative rates decrease with more detailed classifiers, the memory and compute resources increase. This gap is present in DCM [37], OpenSketch [36], and the Resource/Accuracy Tradeoffs in Software-Defined Measurement [26].

2. Filtering relies on thresholds from ground-truth, which may be unknown.

   Anomaly detection and other machine-learning based approaches define a threshold that separates normal from anomalous behavior. Without the ground-truth knowledge of benign behavior, however, setting this threshold is problematic. Being too cautious and setting a low threshold can generate many false positives, while too high of a threshold can cause the system to miss genuinely anomalous events. This gap is present in Kargus [20].

3. Advanced filtering techniques require customized infrastructure.

   In order for filtering to efficiently scale to high line rates, dedicated infrastructure (e.g., firmware or switch TCAM) is necessary to make specific operations computationally inexpensive. Most commonly, parallelized pattern-matching and rapid hash function computation are the most critical operations for filtering. Moreover, switches are not natively capable of any operation other than simple matching and comparison. This gap is present in DCM [37], OpenSketch [36], and the Resource/Accuracy Tradeoffs in Software-Defined Measurement [26].

4. Efficient filters are shallow, while interesting features require deep inspection.

   Efficient filters rely on stateless, rapid analysis of packet headers. Unfortunately, interesting

features often require state to be maintained (e.g., if a connection is outgoing or incoming) or for the packet payload to be inspected (e.g., filtering based on website URL). This gap is present in DCM [37], OpenSketch [36], Kargus [20], and the Resource/Accuracy Tradeoffs in Software-Defined Measurement [26].

5. Filtering does not necessarily reduce storage size.

   When there is no *a priori* knowledge of which data is relevant to collect, filtering data prior to collection may result in throwing away useful information while retaining useless noise. Even when a classifier does exist, the relevance of data may change over time and prompt re-analysis of collected data is necessary. As a result, often the entirety of data must be collected and stored, which reduces some of the benefits of filtering. This gap is present in DCM [37], OpenSketch [36], Kargus [20], and the Resource/Accuracy Tradeoffs in Software-Defined Measurement [26].

The following techniques represent current academic research on data filtering for network knowledge synthesis. While all of these gaps are present in at least one technique, gaps 4 and 5, are present in every technique. Furthermore, gap 2 is not addressed by any technique.

5.3.3 Technique: Distributed Collaborative Monitoring (DCM)

Description of Technique
DCM [37] uses a custom controller and firmware on SDN switches to implement scalable and memory-efficient per-flow monitoring. Each DCM switch uses a two-stage Bloom filter to implement monitoring rules installed on it by the controller. A Bloom filter is a space-efficient probabilistic data structure that represents a set of items and allows rapid membership checking. DCM's first filtering stage is an admission Bloom filter, which represents the set of all flows to be monitored on that switch. If a packet header matches the filter, it is passed to a set of action Bloom filters. Membership in any of these triggers a monitoring action unique to that filter, such as sampling or incrementing a counter.

The DCM controller is responsible for determining, based on network topology and routing state, which flows should be measured on each switch. It load-balances measurement tasks across all switches in order to make maximize measurement accuracy, as Bloom filters have a false positive rate inversely proportional to how much space they are allocated. By distributing tasks across all switches, maximal memory may be allocated to each Bloom filter. The controller also ensures that no flow is measured twice, which prevents oversampling and resource wastage.

Technique-Specific Gaps
DCM is intended to be used alongside SDN architectures like OpenFlow, but it is not clear how its custom firmware and controller compose with such architectures. Additionally, composing measurement and forwarding architectures requires resources (e.g., switch memory) to be split between the two. This introduces a trade-off between measurement accuracy and packet forwarding granularity that is not explored in the paper.

5.3.4 Technique: OpenSketch

Technique Description
OpenSketch [36] provides line-rate streaming analytics for SDNs that supports complex statistic collection and queries via switch-based sketches. A sketch is a compact probabilistic data structure with tunable false positive and negative rates that is able to efficiently count the incidence of specific events (such as the arrival of a packet that is part of a specific flow, has a certain source IP prefix, etc.). OpenSketch's SDN controller component provides an API for building data collection and analysis tools. The necessary switch-level sketches (count-min, bloom filter, etc.) are automatically derived and implemented given the user's queries and an accuracy requirement. The controller optimizes each sketch to use minimal memory by combining modular hashing, classification, and counting primitives.

These three primitives are efficiently implementable in COTS switches using custom firmware. Hashing and classification can be implemented efficiently with minimal usage of TCAM, a high-power, high-speed, parallel form of memory that switches also use for packet routing. Counting uses a memory-efficient dense table stored in SRAM, a lower-performance but much more available form of memory used by switches for noncritical tasks.

Technique-Specific Gaps
OpenSketch requires custom switch firmware to implement on-switch sketches. It is unclear how this would compose with OpenFlow architectures: at minimum, the controller would have to be rewritten to accept the expanded set of control protocol messages. Additionally, resources at switches (e.g., TCAM memory) must now be split between routing and measurement tasks. How to balance these competing goals is not addressed by the paper. Control plane resources (e.g., bandwidth, controller processor utilization), must similarly be divided. No evaluation of this control overhead is provided.

Additionally, OpenSketch is an inherently local solution to data analysis, in that it does not consider the sensor placement problem. All sketches are installed on all switches. This may result in over-sampling of flows and inefficient utilization of switch resources.

Finally, this system is based on streaming analytics using sketches. The measured data is not stored for further analysis, and all analysis tasks must be known in advance and programmed into the controller. Thus, forensic analysis or other attempts to query past network states is not possible using this tool.

5.3.5 Technique: Kargus

Technique Description
High performance IDSs are needed as many networks operate at the rate of tens of billions of bits per second. However, hardware-based approaches to high performance IDSs do not allow for operational flexibility and are costly. Software-based techniques allow for new patterns to be added and are less costly, but are not yet ready for high-speed networks [20].

As such, Kargus was developed to solve that need. Kargus is a software based IDS that utilizes modern hardware components to achieve the performance necessary for high-speed networks. It does so via two techniques: (1) applying batch processing from packet reception to pattern matching, and (2) using high process parallelism to balance the load of flow processing and pattern matching across CPUs and GPUs (graphic processing units) [20].

Technique-Specific Gaps

Kargus' performance is highly dependent on packet size. As packet sizes get larger, there is a reduced benefit of batch processing and increased overhead. As packet sizes get smaller and the attack portion increases, there is also degraded performance as more cores are needed for pattern matching. In addition, there is a performance degradation caused by the consumption of extra CPU cycles when attack signatures span multiple packets in the same flow. Lastly, the use of GPUs for offloading has limitations within itself, i.e., offloading to GPUs requires additional CPU usage and memory resources for data preparation, copying, etc., and GPUs consume more power than CPUs [20].

5.3.6 Technique: Resource/Accuracy Tradeoffs in Software-Defined Measurement

Technique Description

This paper [26] investigates the performance of three primitives used in streaming analytics of network traffic: hashing, counters, and arbitrary code fragments. Each of these can be implemented in switch firmware and configured by a centralized controller, much like SDN architecture but focused on measurement rather than packet forwarding. These primitives can be combined with one another to build a variety of switch-based traffic sensors with a cost in switch memory, switch processing, and communications bandwidth to the controller.

Hashing primitives use high-speed but low availability TCAM memory to extract sketches or summaries of traffic at switches and transfer these higher-level results to the controller. They can easily be reconfigured by the controller to change what is being measured, and can measure a wide variety of traffic with a small number of hash functions. This makes hashing very memory-efficient and suitable for highly variable traffic patterns. Since analysis and configuration is off-switch, however, the timescales in which hashing-based sensors can be effectively used are on the order of seconds due to control loop delay.

Flow counter primitives, such as those in OpenFlow, increment byte or packet counts whenever their associated flow rule is triggered. These counters are generally limited to wildcarded packet header matches and thus do not scale well to highly variable traffic in terms of switch-memory usage. Analysis is additionally outsourced to the controller, which must poll the switch periodically. This limits counters to analysis timescales of seconds and measurement of fairly stable traffic patterns.

Finally, on-switch code execution of simple programs can offload some analysis tasks to switches and allow more advanced data collection and preprocessing. However, this utility comes at the cost of much higher switch processing and power costs. Code execution can be used for very

fine-grained measurement (on the order of milliseconds) of highly variable traffic due to the lack of a control loop.

Technique-Specific Gaps
The paper provides a generic qualitative analysis of hashing, counting, and code execution. Quantitative analysis is limited, however, to varying implementations of the hierarchical heavy-hitters problem over an Internet trace dataset. It is unclear if the paper's various implementations are optimal, or how other analytics or sensors (e.g., Bloom filters) would perform. Furthermore, Internet trace traffic is substantially different from traffic in datacenter and enterprise networks. Whether the conclusions drawn by this paper extend to those environments unchanged is not addressed.

Additionally, this work only considers local sketches on a single switch. It does not address a network of multiple switches, how sensors would be placed globally, or how their measurements can be composed into a single view of network state.

5.4 GAPS IN DATA ANALYSIS

5.4.1 High-Level Gaps

1. Requires thresholds for anomaly detection, which are manual, can be evaded, and need ground truth
2. Often evaluated in a closed world with no noise
3. Network conditions change faster than analysis convergence
4. Complex analytics are not scalable
5. Window-based analysis is limited as large windows are unscalable, but small windows are easy to evade
6. Often requires out-of-band information
7. Small packets and fragmentation degrade performance
8. Relies on higher-level identities, e.g., user ID instead of IP address
9. Assumes a benign environment and is vulnerable to poisoning
10. Often assumes the majority of devices are healthy
11. More effective techniques are active, visible to attackers, and perturb the network

5.4.2 Description of Gaps

1. Requires thresholds for anomaly detection, which are manual, can be evaded, and need ground truth.

   Anomaly detection approaches define a threshold that separates normal from anomalous behavior. Without ground-truth knowledge of benign behavior, setting this threshold is problematic. A low threshold can generate many false positives, while a high threshold can cause the system to miss genuinely anomalous events. Additionally, if the value is known to an attacker, then it can be evaded either by using low-rate attacks to trigger a false negative, or by creating distracting storms of false positives. This gap is present in Blacksheep [6], ALERT-ID [7], Detecting Distributed Brute Forcing [21], Beehive [34], ANAX [3], and Mimicus [32].

2. Often evaluated in a closed world with no noise.

   Many analysis techniques rely on training a classifier to recognize abnormalities in data. Unfortunately, their evaluation frequently uses a closed-world environment with no noise or perturbations to the dataset that could be expected in a production environment. This

makes it difficult to judge their genuine efficacy. This gap is present in Host Fingerprinting and Tracking on the Web [35] and Mimicus [32].

3. Network conditions change faster than analysis convergence.

   When analysis is performed offline, or online but with a high time to convergence, the conclusions drawn from that analysis may no longer hold with respect to the current network state. This is especially difficult in highly dynamic networks with many users coming and going, a large volume of short-duration network flows, etc. This gap is present in ALERT-ID [7], Input Framework [1], and Distilling Critical Attack Graph Surfaces [19].

4. Complex analytics are not scalable.

   Complex analytics (e.g., hierarchical heavy-hitters, traffic analysis, and application-layer analysis) often require keeping substantial state, inspecting packet payloads, or using algorithms that have resource requirements that scale super-linearly with the size of the dataset. None of these requirements scale to large network sizes or highly dynamic traffic conditions. This gap is present in Count Me In [2], Input Framework [1], FRESCO [30], Distilling Critical Attack Graph Surfaces [19], Trail of Bytes [24], and Blacksheep [6].

5. Window-based analysis is limited as large windows are unscalable, but small windows are easy to evade.

   Keeping a history of network state is necessary when current conditions may or may not be anomalous given prior events. In order to scale, however, analytics cannot keep a complete history of the network and must make do with a window extending some duration into the past. Large windows require infeasible amounts of storage. Small windows, however, allow an adversary to mount low-rate attacks where each stage arrives after the system has 'forgotten' the attacker's previous actions. This gap is present in Count Me In [2], Input Framework [1], Detecting Distributed Brute Forcing [21], and ANAX [3].

6. Often requires out-of-band information.

   Detection of anomalous events often relies on access to out-of-band information like blacklists of malicious websites or hand-labeled training data. These must be manually updated, configured, and maintained. The analytic is often only as good as the out-of-band information it relies on, making evaluation difficult. This gap is present in Input Framework [1], Blacksheep [6], Host Fingerprinting and Tracking on the Web [35], Detecting Distributed Brute Forcing [21], ANAX [3], Trail of Bytes [24], and Mimicus [32].

7. Small packets and fragmentation degrade performance.

   Most analytics only inspect packet headers, not payloads. This makes networks with many small packets much more performance-intensive to analyze than networks with fewer, higher-volume packets. Additionally, when large packets are fragmented into a series of smaller packets, all fragments must be stored and re-assembled before analysis is possible. This

gap is present in Input Framework [1], Host Fingerprinting and Tracking on the Web [35], Detecting Distributed Brute Forcing [21], and ANAX [3].

8. Relies on higher-level identities, e.g., user ID instead of IP address.

   Application-layer analytics (e.g., those that parse log files from servers) have visibility into user names and high-level identities. Mapping these to analytics that observe network traffic is difficult, as IP addresses may change frequently while user identifiers are fairly static. This gap is present in ALERT-ID [7], Beehive [34], and Distilling Critical Attack Graph Surfaces [19].

9. Assumes a benign environment and is vulnerable to poisoning.

   Despite the fact that attackers have been poisoning network control data for decades (e.g., ARP cache corruption or rogue DHCP servers), many analytics implicitly trust the data that they are given. This can allow an attacker to evade detection or cause mis-attribution of the attack to an innocent victim. This gap is present in ALERT-ID [7], Input Framework [1], Count Me In [2], Beehive [34], ANAX [3], Trail of Bytes [24], and Mimicus [32].

10. Often assumes the majority of devices are healthy.

    Analytics which compare the behavior of one entity to a group of similar entities assume that the larger group is implicitly healthy or benign, or that at least a majority of the entities are benign. For example, a behavioral analysis may compare an end-host's behavior with that of other end hosts sharing the same operating system. In cases where the group of similar devices is of small size (e.g., machines running Linux), an attacker could potentially cause the majority to become defective and induce the analytic to falsely flag the benign hosts as malicious. This gap is present in Blacksheep [6] and Beehive [34].

11. More effective techniques are active, visible to attackers, and perturb the network.

    An analytic technique can often be made more effective by allowing it to probe traffic of interest and observe the behavior. However, this perturbation can also be seen by attackers that can change their behavior and learn to evade detection. Additionally, probing may cause benign traffic to experience delays or corruption, leading to accidental denial off service. This gap is present in Trail of Bytes [24] and Blacksheep [6].

The following techniques represent cutting-edge academic research on data analysis for network knowledge synthesis. The most common gaps are 1, 4, 6, and 9, which are present in approximately half of the techniques. Furthermore, gaps 2, 7, 9, and 10 are not addressed by any technique.

5.4.3 Technique: Blacksheep

Description of Technique
Malicious rootkits are a type of software that is designed to modify an operating system without

detection. Often, infections begin on a single machine and spread to a larger portion of the enterprise through kernel-based rootkits. Rootkits' ability to evade detection has remained an unsolved problem as only 7-10% of all anti-virus protections are focused on rootkits [6].

Blacksheep was created to solve this problem by detecting stealthy kernel rootkits. Blacksheep provides an advantage over other detection techniques, as it does not rely on signatures for detection. Instead, it leverages the fact that organizations use standard software and configuration settings for most machines. In this way, both zero-day vulnerabilities and long-known threats can be identified by looking for anomalies among groups of similar machines [6].

Blacksheep achieves this goal by first acquiring memory from computers with identical (or similar) hardware and software configurations. (Memory dumps are obtained using one of four methods: (1) software acquisition, (2) crash dumps, (3) hardware acquisition, i.e., through external peripherals, or (4) virtual machine introspection.) Next, the memory dumps are transferred over the network to the Blacksheep analysis server. From there, the server compares pairs of memory dumps to compute a distance metric that describes the difference between the two memory dumps. Distance metrics are calculated for four entities: (1) high-level configuration differences across kernel modules, (2) driver code differences, (3) kernel data differences, and (4) kernel entry point differences. Next, all four distance metrics are normalized. The clustering engine uses the aggregated distance metric to generate hierarchical clusters. Infections are identified based upon cluster size, e.g., if a cluster only contains one computer, then it is infected [6].

Technique-Specific Gaps
The reliance on many machines for analysis leads to several limitations within Blacksheep. In particular, Blacksheep assumes that the majority of machines (90%) are benign, i.e., not infected. If the majority of machines are infected, the clustering technique will yield large clusters of infected machines and small clusters of noninfected machines, thereby defeating its own technique. Also related to the use of many machines is the requirement that the machines are similar across hardware and software configurations and that the memory dumps be obtained when the machines are in a similar state. This is a stringent requirement that may not occur frequently in enterprise systems where users are allowed to customize their machines [6].

Blacksheep requires a viable data dump for analysis. More specifically, Blacksheep works best when memory dumps are obtained from virtual machine introspection. The reliance on virtual machine introspection for memory dumps is problematic if most computers within the enterprise do not use virtual machines. Outside of memory dumps, the actual analytical approach used within Blacksheep requires manual analysis with respect to cluster formation. Lastly, stealthy attackers can evade Blacksheep's techniques by modifying parts of the kernel memory that change frequently or by tampering with the method in which memory dumps are acquired, e.g., a rootkit could use virtual machine detection techniques to modify or terminate its behavior [6].

5.4.4 Technique: ALERT-ID

Description of Technique
Securing network infrastructure can be challenging in a large enterprise network. A common prac-

tice in securing a network is to deploy periphery and centralized authentication and authorization schemes. However, the switches and routers in these networks are widely dispersed and managed by a large group of network operators. Because of the large number of operators and periphery devices in an enterprise network, there is the potential for misconfiguration, which can render the network ineffective against attacks.

The authors present an intrusion detection system [7] that is built on existing authentication and authorization framework. The purpose of the system is to gather router and switch access logs in real time so as to provide a global view of the network. To construct the anomaly detection system, the authors analyze logs collected over six months from the Terminal Access Controller Access-Control System Plus (TACACS+). The authors identify features based on device access patterns such as stability of router control commands and configuration, and users' behaviors such as login attempts and login access patterns. A model is constructed using these features from the six-month data from the TACACS+ logs. The model is then used to detect anomalies (deviation from normal behavior) in real time.

The authors evaluate their system by performing experiments and determine that their system identifies potential intrusions with an acceptable level of false positives.

Technique-Specific Gaps
Given that ALERT-ID relies on the log files of another system, it makes it system vulnerable to logs that have been manipulated by an attacker. Without a realistic ground truth, it is difficult to determine the efficacy of ALERT-ID. With the massive amounts of data flowing through networks, it is difficult to determine how fast ALERT-ID is in processing large amounts of data.

5.4.5 Technique: Input Framework

Description of Technique
There are a variety of efforts dedicated to providing network-security information such as blacklisted websites from Google's Safebrowsing URL list and Virus Total's hash-based malware identification. Some organizations produce much more context-rich network-incident information such as data from REN-ISAC's security event system or Department of Energy's Joint Cybersecurity Coordination Center (JC3) feeds. However, current intrusion detection systems (IDS) focus primarily on analyzing packet information and cannot readily incorporate these external information into their analysis. Not being able to incorporate external information into the analyses of IDS limits the capability of an IDS.

The authors present Input Framework [1], which is able to integrate information from external sources into the IDS decision process in real time. The Input Framework is built on the Bro IDS platform [28]. To enable Input Framework to achieve its goal of incorporating external information source in real time into an IDS, it must meet a number of objectives. The objectives are: (1) it should be adaptable to different sources of information, (2) provide a simple and flexible user interface, (3) provide asynchronous I/O, and (4) perform operations in real time.

The Input Framework consists of a manager that interfaces between Bro's core and the external information sources. When the Input Framework receives a request to open a stream, it spawns a reader instance that connects to the corresponding information source. There are different kinds of readers: file readers, database readers, etc. The readers forward the external information to the manager, which passes it on to the IDS analysis engine. Note that the readers are spawned asynchronously.

In evaluating their system, the authors deploy it at Lawrence Berkeley National Laboratory (LBNL). LBNL prefers using SES and JC3 feeds and imports them directly into Input Framework. The Framework was deployed for two months, during which it was able to improve the security posture of LBNL. In terms of latency, the Input Framework does not add significant delays when receiving external information from a source.

Technique-Specific Gaps
Although Input Framework's capability extends beyond a normal IDS system, it still has some gaps. The Input Framework is suppose to analyze external information in real time; however, there is no evidence showing that it can process large amounts of data in real time. The Input Framework can also be manipulated by an adversary that is able to poison information from external sources, which can reduce the efficacy of an IDS. Finally, the Input Framework is susceptible to all the limitations of traditional IDS.

5.4.6 Technique: FRESCO

Description of Technique
FRESCO [30] provides a modular security monitoring and analysis application development platform for OpenFlow SDNs. Apps are written in a simple controller-agnostic scripting language that defines five kinds of interfaces: *Inputs* receive values from other apps or the controller; *Outputs* transmit values to other apps or the controller; *Parameters* are module-specific configuration values; *Events* register the module for triggering when that event occurs; and *Actions* implement operations on flows or packets.

Individual apps can be linked together in a circuit to make larger applications, by specifying connections between input and output interfaces of different apps. FRESCO also has a controller-specific security enforcement kernel that manages rule conflicts and controller-specific implementation of OpenFlow events and sensors. Currently it is only written for NOX.

Example applications are provided, which include network anomaly detectors, attack detection tools, malware detectors, and a honeypot-based quarantine system. The overhead for running FRESCO scripts (in terms of flow setup time) varies by script from 0.5 to 15 ms in several evaluated example applications.

Technique-Specific Gaps
FRESCO may not scale to deployments with frequent network events or updates. The evaluation section is very light, but the reported overhead of running example scripts is up to 15 ms on a

network with only three hosts. Additionally, while FRESCO's scripting language is highly modular, it may not be very readable or easy to maintain.

Finally, the FRESCO SEK is only implemented in NOX. This is a C++-based controller that is not well-maintained and does not support modern versions of the OpenFlow specification.

5.4.7 Technique: Host Fingerprinting and Tracking on the Web

Description of Technique
This paper [35] analyzes month-long Hotmail webmail and Bing search query datasets. The authors find that with only user-agent strings, users can be identified with 63% precision and 72% recall. When coarse-grained 24 IP prefixes are also used, precision and recall improve to 79% and 69%, respectively. This is comparable to the efficacy of cookies (but without the need to store state on clients), which scored 83% precision and 68% recall.

The authors investigate the unexpectedly low efficacy of cookies in uniquely identifying users. The primary cause is ascribed to cookie churn, which is defined as a cookie ID appearing once in a dataset but never a subsequent time. In the Search dataset, the churn rate was 48%. By combining cookie IDs with browser UA strings and IP prefixes, however, 88% of users who clear cookies between sessions can still be identified.

The authors also study host mobility, which is defined as the changing of a host's IP address on both fine-grained (DHCP) and coarse-grained (inter-domain) scales. Hosts are identified as having moved by having the same cookie id. By analyzing inter-domain host mobility patterns, the authors detected (without initially looking for) cookie-forwarding attacks via anomalous incidences of host mobility from known IP ranges to domains that are either exclusively sinks/sources (i.e., hosts move only into or out of the domain) or domains not associated with end-host traffic.

Technique-Specific Gaps
The paper studies certain features of end-host traffic accessible to web-servers (IP address, user agent string, browser cookie, and user login ID), but provides minimal explanation of what other features are available or how they might be utilized to more uniquely fingerprint end-hosts. Similarly, the technique is focused only on web servers and how they can fingerprint end-hosts. Other classes of Internet-facing servers (IRC, VoIP, DNS, etc.) are not considered.

5.4.8 Technique: Measuring the Similarity of Network Hosts

Description of Technique
This paper [9] presents a technique to unify disparate types of network data into a single space to facilitate comparison of hosts via a distance metric. The technique proceeds as follows: First, it creates a database of all network data where columns denote measured values and are tagged as either numeric or categorical. Second, it gives numeric spaces (i.e., those whose semantics of similarity map to distance on the real number line) a simple linear difference metric. Third, it gives categorical spaces a hierarchy-based distance metric, where distance is based on the level of two points' common ancestor. Fourth, it combines all metric spaces into one via a custom product and

normalization function. Standard techniques for renormalization either destroy semantic content or over/under value spaces due to differences in measurement scale. This hybrid piecewise approach maps all spaces sharing the same units of measure (such as seconds, minutes, hours, and days) to constant ranges on the [0,1] interval such that the normalized differences represent the relative severity/amplitude of each unit.

Finally, to compare temporal distances, hosts are represented as the time series of their events. To make the technique tractable, an adapted heuristic version of Dynamic Time Warping (a well-known technique in the speech recognition community) is applied prior to time-series analysis. The technique uses dynamic programming and an adapted version of the Sako-Chiba heuristic (which evaluates only matrix cells close to the diagonal) to run time warping in linear time, compared to the naive algorithm, which requires quadratic time.

Technique-Specific Gaps
The technique's accuracy is only compared to the L1 distance metric. No discussion of other, more domain-specific metrics is provided. This makes it difficult to truly judge its efficacy.

Additionally, while host classification is cited as a use case, the authors neglect to discuss classic, well-known techniques to fingerprint host operating systems and applications based on network traffic and probe responses (such as nmap scans).

Finally, the example metrics defined in the paper are very simple (e.g., port distance metrics based on whether two ports are well-known, registered, or ephemeral). While the authors allude to more complex metrics, it is not clear how metric complexity impacts performance or how some metrics would even be constructed using their hierarchical approach (e.g., port relationships that are application-specific).

5.4.9 Technique: Count Me In

Description of Technique
Researchers rely on statistical properties of network data for traffic profiling and control. The statistical measurement of these properties are used to provide insight into network monitoring and security issues. However, current techniques do not provide flexibility in the computation of different kinds of network-data statistics.

The authors present a novel framework [2] for computing different kinds of summary statistics in real-time, independent of the underlying data that is potentially collected from multiple monitoring points. The objectives of the summary statistics framework are: (1) provide a simple and flexible user interface, (2) be data agnostic, i.e., operate on different kinds of features, (3) be extensible, i.e., support different statistical computations, (4) process and compute statistics in real time, and (5) scale to large networks.

The framework works by processing network-traffic data in real time. As the framework continuously receives the data, a reducer aggregates the results. In general, reducers compute statistical measures on the data that they process. Some of the statistical measures that are amenable to real-time computation are summation, average, deviation, variance, minimum, and

maximum. The framework also supports the distributed computation of the statistical measures when data is received from multiple network sources.

The authors implement their framework on the Bro network monitoring platform [28]. The authors test their summary statistics framework by developing a script for detecting address and port scans, brute-force login detection, SQL injection detection, and trace-route detection. They compare their framework to the Bro platform and demonstrate that their framework has a mean memory overhead of 6.7%.

Technique-Specific Gaps
Currently, the framework is able to compute simple statistical measures in real time. However, it is not known how the framework handles more complex measures in real time. The statistical measures computed by the framework are also susceptible to noise in the network data, especially in the distributed setting. It is not clear how the authors address the problem of noise in the data. Also, it is not clear how the technique will scale to large amounts of data. Although having a memory overhead of 6.7% may be acceptable in the experimental setting, the memory overhead can be significant in the presence of complex statistical measures and large data.

5.4.10 Technique: Detecting Distributed Brute Forcing

Description of Technique
With respect to brute-forcing attacks, adversaries can distribute their efforts across multiple sources to evade detection. This is particularly successful as most detection mechanisms set thresholds assuming that only individual machines participate in brute-force activity [21].

A solution to the problem of distributed brute-force attacks relies on detection through the analysis of aggregated activity. More specifically, the distribution of login failure rates occurring across the site is examined. Based upon operator-defined parameters that describe the desired false alarm rate and detection characteristics, change point detection is used to identify significant changes in the distribution. Next, the relevant attack epoch is defined. Each remote host that is present during the attack epoch is classified as either legitimate, e.g., an actual user who failed to log in, single brute force, or distributed brute force based upon the host's past history and commonalities with respect to servers and usernames [21].

Technique-Specific Gaps
Detecting brute-force activity is a novel concept, but its current implementation has a few limitations. The differentiation between legitimate, single brute force, and distributed brute force is done manually. In addition, the performance of the system was calculated based upon limited to no ground truth data. In essence, the effectiveness of the system is still unknown. Lastly, many assumptions were made concerning the definition of a brute-force attack that do not always hold true:

- More than 20 failed login attempts indicates a brute forcer; this allows for brute forces that probe at low rates to be missed

- If 19 unsuccessful attempts are followed by a successful attempt, then the activity is classified as legitimate; this allows for successful brute forcers to be misclassified as legitimate [21]

5.4.11 Technique: Beehive

Technique Description
As attacks on computer networks increase, organizations are deploying more security technologies from different vendors to mitigate these attacks. These security technologies generate large amounts (e.g., terabytes) of situational-awareness data about the state of the network, which are stored in log files in different formats. However, due to the large size and different log formats, the analysis of the data becomes intractable.

The authors present an automated approach to analyzing the log files, called Beehive [34]. The threat model for Beehive is to detect network hosts that are infected and host-based user activities that violate enterprise policies. To build Beehive, the authors had to address the following challenges: (1) how to analyze large log files; (2) how to identify meaningful security incidents from large log files given that there is a significant semantic gap between the logs collected and what security analysts require to identify suspicious host behavior; and (3) how to address the lack of ground truth for evaluating Beehive.

Before Beehive begins its behavioral analysis, it first preprocesses the log data. There are four steps in the preprocessing phase: (1) timestamp normalization, (2) IP address-to-host mapping, (3) static IP address detection, and (4) identification of dedicated hosts. After preprocessing the data, Beehive performs feature extraction. There are four classes of features extracted by Beehive: destination, host, enterprise policy, and traffic-based features. Beehive performs clustering using the features as there is no ground truth data. Top outliers that occur after clustering are reported to the security analyst.

Beehive was able to detect network infections that violated malware or security policies. These violations were not detected by antivirus software or by their enterprise security operations center. Out of 784 incidents generated over a two-week period, 1.02% of the incidents were identified by state-of-the-art security tools, whereas Beehive was able to detect and categorize the various violations: malware-related (25.25%), business policy violations (39.41%), and unrecognized, but automated, software and services (35.33%).

Technique-Specific Gaps
The dependence on clustering because of the lack of ground-truth data can slow the process of automatically identifying suspicious incidents. There is the potential for a high number of false positives, which can overwhelm security analysts and cause them to stop using Beehive. Because of the lack of ground-truth data, it is not clear whether the features employed by Beehive are adequate to detect real malicious incidents.

5.4.12 Technique: Trail of Bytes

Technique Description
Current mechanisms for forensic analysis either rely on code that is resident in kernel space or make changes to application binary interfaces. Unfortunately, these methods are subject to tampering, e.g., attackers can inject code into the kernel space, thereby changing log integrity. Virtualization provides an alternative to this situation as code and applications can be sandboxed [24].

A particular example of a technique that leverages virtualization for forensic analysis is the Trail of Bytes technique. It begins by monitoring specific virtual machine blocks that are on a watchlist to see if they are accessed by syscalls. Once a block on the watchlist is accessed, a timestamp is recorded and a signature for the code page is created and stored. The physical page where the blocks are paged-in is also monitored. The parameters of the syscalls are parsed if they are file system objects, memory resident objects, shared memory objects, or network objects. To infer data movement, the source and destination of the parameters is also examined. After all reads/writes to the blocks are completed, an entry to the audit log is added [24].

Technique-Specific Gaps
The Trail of Bytes technique suffers from a few limitations. For one, the technique is only for virtual machines, and as such, it is prone to hypervisor-detection attacks. On top of hypervisor-detection attacks, this technique can be evaded by attackers through resource exhaustion attacks. This technique also extends the trusted code base, e.g., if the security of the hypervisor is compromised, so is the security of the system. Lastly, it only monitors and logs events from blocks on a watchlist; it does not examine blocks that are not on the watchlist, nor does it prevent attacks [24].

5.4.13 Technique: NetPlumber

Technique Description
NetPlumber [22] is a data analysis tool for real-time time checking of network policies (i.e., properties that should always hold) on SDNs. Example policies include the absence of forwarding loops, waypointing (traffic from A to B should always pass through C), or time-gating (web traffic is allowed only during business hours). Every time a new OpenFlow rule would be installed, or an existing rule changed, the system first checks to ensure that doing so would not lead to violation of any invariants.

In order to perform policy checking in real time and avoid network slowdowns, NetPlumber uses an augmented version of Header Space Analysis (HSA) [23]. HSA uses a generic, geometric interpretation of packet headers to compute reachability sets: the set of all packet headers that can be sent from each end host to each end host. NetPlumber extends this with an incremental update mechanism based on packet header equivalence sets. Using this incremental technique, HSA checks can be performed in near real time.

NetPlumber is evaluated on large-scale networks, including the Stanford University network (mid-size enterprise) and Google WAN, one of the largest SDNs known to be deployed. Even at the largest scales, all-pairs reachability policies can be checked in 0.1 to 10ms. The complexity

of policy-checking in general is linear in network size and the number of policies. A distributed variant is also provided and that can reduce this complexity by parallelizing many of the checks.

Technique-Specific Gaps
NetPlumber's policy language does not allow arbitrary policies. It is based on regular expressions and logical operations, with constraints defined over either packet headers or paths taken through the network. Thus, complex policies that cannot be expressed as a regular expression cannot be checked in NetPlumber. Examples of such unsupported policies are those that are self-referential, stateful, or rely on evaluation of arbitrary code fragments.

5.4.14 Technique: Distilling Critical Attack Graph Surfaces

Technique Description
Generating attack graphs for enterprise networks can result in large complex graphs that are difficult to analyze by system administrators without the right visualization. Also, computing quantitative risk assessments over the large graph can result in substantial performance penalties. However, computing risk assessments may not require the whole graph, but rather a critical-attack-graph surface, where a critical attack graph is a subgraph of the entire attack graph that contains the most critical security problems in a system.

The authors present an attack-graph-distillation approach [19] that enables the user to control the amount of information presented to them in an attack graph. For example, the user can choose to view only the critical attack paths that an attacker can use to compromise a given host based on some severity metric.

To generate critical attack graphs, the authors first transform the full attack graph into a CNF Boolean formula ϕ. Transforming the attack graph into a Boolean formula enables more flexible analysis because a system administrator can augment the Boolean formula, ϕ, with their own constraint-based logic requirements. The authors use CVSS access complexity metrics to derive the attacker's cost metric for literals in the resulting Boolean formula. The resulting constrained and cost-related formula becomes the initial SAT problem, which is input into a MinCostSAT solver. The goal of the MinCostSAT solver is to answer the question "What is the next critical attack path?" on each iteration. For each iteration of the SAT problem, a solution resulting in a realistic attack graph path is generated or counter-example constraints are generated, which are then generalized into lemmas that are used to augment ϕ. The Boolean Satisfiability problem (SAT) is incrementally updated with lemmas learned from the previous iteration. SAT refinement using the CounterExample-Guided Abstraction and Refinement (CEGAR) approach removes spurious paths from the SAT. For example, a solution may result in cycles, unrealistic paths, or zero-cost nodes present in the attack paths. These issues are handled by a solution refiner and verifier. The critical attack graph resulting from the final solution to the SAT problem consists of only the literals that are *True*.

The authors evaluate their technique in the context of an enterprise network with thousands of nodes. The authors show that to distill a critical attack graph with about 1000 real attack paths from 40000 and 20000 CNF clauses, their technique took 5500 and 3000 seconds, respectively. They

also showed that their technique is accurate in capturing critical security problems with just 15% of the original attack graph as the critical attack graph.

Technique-Specific Gaps
Although their technique is effective in generating critical attack graphs, it is not known how many constraints are present in a real enterprise network. Also, it is not clear

- how accurate and consistent the CVSS complexity metrics are when assigned to the literals in the Boolean formula
- how the technique can handle dynamically changing networks
- whether critical attack graphs are contained in 15% of an entire attack graph in a real enterprise network.

5.4.15 Technique: ANAX

Technique Description
The Domain Name System (DNS) is an important component of the Internet backbone. It enables mapping between domain names to IP addresses and other records essential for email, web services, and other network protocols. However, DNS is susceptible to poisoning attacks. A poisoning attack allows an adversary to corrupt DNS records thereby enabling the adversary to manipulate resolution caches. For example, through DNS cache poisoning, an attacker will be able to assign wrong IP addresses to domain names. Approaches such as DNSSEC and DNSCurve have been proposed, but adoption has been slow because of the infrastructure changes that need to be made. Intrusion Detection Systems (IDS) are also not capable of detecting such poisoning attacks.

The authors present a DNS protection system called Anax [3] that is able to detect Kaminsky-style DNS cache poisoning. The intuition behind Anax is that DNS records generally direct users to a known and usually stable set of NS records. Poisonings redirect users to new and different IP addresses often set up for short periods of time. Anax also relies on the fact that poisoned open-recursive DNS (ORDNS) will report cached resource records that are abnormal in terms of zone and the IP address space.

Anax's detection has three discrete phases: preparation, measurement, and analysis. In the preparation phase, Anax collects IP addresses of ORDNS around the world, determines which domains are susceptible to poisoning attacks, and probes the ORDNS to detect poisoning. In the measurement phase, Anax's scanning engine conducts a number of DNS queries and records the matching answers. It stores the raw DNS traffic in an indexed database. In the analysis phase, Anax checks the stored resource records and determines whether the recorded resource is legitimate using a trained machine-learning classifier.

The authors conducted experiments showing that Anax was effective in detecting Kaminsky-style attacks and produced a false positive rate of 0.6% and a true positive rate of 91.9%.

Technique-Specific Gaps
Anax requires manual labeling of network data to build the machine-learning classifiers, which can be tedious and time-consuming. There is also an opportunity to mislabel the data. Anax maintains a whitelist of benign DNS records. However, maintaining a whitelist can be tedious and prone to errors. Anax uses passive DNS data when computing statistical features, which makes it sensitive to the relative DNS window and how the passive data are aggregated. The use of past CDN IP address space for poisoning could also be a significant evasion threat for Anax if the passive DNS window is more than a few weeks old.

5.4.16 Technique: Mimicus

Technique Description
For malware analysis, machine learning tools allow for automatic attribution. Machine learning can also be used to detect security violations, e.g., drive-by-downloads. When testing these machine learning approaches, it is often the assumption that the adversary has no knowledge of the analysis approach. At the same time, most publications demonstrating the ineffectiveness of machine learning techniques assume that the adversary has full knowledge of the analysis approach. What is needed is an evaluation of machine learning tools that is more representative of attacks in the wild, i.e., adversaries who have limited knowledge of the analysis approach [32].

Mimicus is an evasion framework that was designed to test the effectiveness of a real, deployed system–PDFrate–that was designed to classify PDF files as benign or malicious. As PDFrate is a research system with documented methodological and technical details, an adversary can gather sufficient information to evade PDFrate's detection capabilities. Mimicus tested the effectivenss of PDFrate by varying how much knowledge an adversary has according to three components: feature set, training dataset, and the classification algorithm [32].

Technique-Specific Gaps
Through Mimicus, many limitations of PDFrate were found. First, if there is any knowledge regarding the feature set that PDFrate uses, then PDFrate's score drops from 100% to 40%, thereby allowing more adversaries to go undetected. If adversaries have knowledge of the training dataset or the classification algorithm, the score drops to 28%. PDFrate runs regular expressions on raw bytes to extract features. This technique was chosen based on its speed, not its quality, thereby allowing adversaries to hide features in encoded or compressed files. PDFrate is trained on a static dataset, with no retraining. If adversaries employ a technique that is newer, and hence not represented in the training dataset, then they will have successfully evaded PDFrate. Lastly, the semantic gap between how PDFrate parses files compared to how PDF readers parse files allows for an attacker to inject arbitrary content into the PDF, thereby poisoning feature extraction that is dependent upon regular expressions [32].

5.5 GAPS IN INFORMATION SHARING

5.5.1 High-Level Gaps

1. Anonymizing data is hard as structure must be preserved and identifying information removed
2. Attack campaigns have different manifestations in different environments
3. Open-source data is over-represented due to availability
4. Real-time sharing is very resource-intensive
5. Biased towards security-conscious organizations

5.5.2 Description of Gaps

1. Anonymizing data is hard as structure must be preserved and identifying information removed.

   Complete anonymization of data while retaining the ability to truthfully answer queries is a major open problem. Any structure (e.g. IP subnets, operating systems, etc.) that is preserved could be used in conjunction with an auxiliary dataset to de-anonymize the original data. This gap is present in Differentially-Private Network Trace Analysis [25] and Provable De-Anonymization of Large Datasets with Sparse Dimensions [11].

2. Attack campaigns have different manifestations in different environments.

   Attacker methodologies are influenced by the context in which an attack is launched. While one organization may suffer service outage as a result of an attack, another might experience anomalous login attempts as a result of the same attack. The artifacts and observable effects of an attack depend on the environment in which they manifest themselves and can vary from enterprise to enterprise. Thus, sharing the artifacts and observable effects may not be informative. Effective information sharing frameworks should share "knowledge" rather than raw data indicating artifacts.

3. Open-source data is over-represented due to availability.

   Due to their sensitive nature, very few public datasets exist that contain network traces or server logs during an attack. Those few that are available thus become over-represented and *appear* indicative of the larger space of attacks. This can lead to conclusions that appear generalizable but which, for example, actually only work in campus networks and not enterprise environments. This gap is present in Firmware Image Analysis [8].

4. Real-time sharing is very resource-intensive.

   In general, sharing copious amounts of sensitive information is difficult. This is exacerbated

when real-time sharing of data with a third party is needed, such as during attack remediation, cleaning, and recovery. For example, cryptographic anonymization techniques are often computationally intensive and require substantial resources from all involved parties. This gap is present in Differentially-Private Network Trace Analysis [25] and Policy-Enhanced Private Set Intersection [31].

5. Biased towards security-conscious organizations.

   Due to the nature of information on cyberattacks, most organizations which produce or consume these datasets are more security-conscious than average. This may make conclusions less generalizable. Additionally, information sharing techniques may require a high level of technical proficiency that, while not uncommon in the cybersecurity space, is nonetheless rare overall. This gap is present in Differentially-Private Network Trace Analysis [25] and Policy-Enhanced Private Set Intersection [31].

The following techniques represent cutting-edge academic research on information sharing both in general, and for network knowledge synthesis. Gap 2 is neither present in nor addressed by any of these techniques, potentially due to lack of available public datasets for study. Gaps 3 and 5 are also not addressed in academic research.

5.5.3 Technique: Firmware Image Analysis

Description of Technique
Embedded systems are present in most devices of an electronic nature, e.g., wireless routers, computers, printers, etc. On each of these embedded devices is special software, known as firmware. Firmware is similar to traditional software in that it can contain bugs and vulnerabilities. Current efforts to analyze firmware images rely on manual analysis of a small subset of devices, thereby yielding non generalizable results [8].

A solution to this problem relies on collecting data from many firmware images and running automated analysis. More specifically, firmware images are obtained from web crawlers and a web interface where security researchers can submit their own firmware images for analysis, e.g., firmware images that need to be manually extracted from a device. Next, the firmware image is unzipped and statically analyzed. During the analysis phase, brute forcing is used to extract password hashes. The firmware images are then correlated based upon four dimensions: shared credentials, shared self-signed certificates, common keywords, and fuzzy hashes. Lastly, the firmware dataset is enhanced by performing online scans and lookup queries, e.g., to find the severity of a vulnerability that is present on multiple devices. This last step can also include finding other firmware images that contain that vulnerability, which leads to sharing of information [8].

Technique-Specific Gaps
While ambitious, the current effort's attempt at analyzing embedded firmware images needs to be optimized. Given that a very small percentage of embedded images is actually obtained from the web interface, the analysis tends to rely on images taken from web crawlers. As such, the analysis

is biased towards open-source data that can be obtained from well-organized vendors, e.g., vendors that properly label their firmware images. In addition, of the firmware images that were collected from web crawlers, only 34% actually contained valid firmware images. This figure (34%) is an estimate as to the robustness of the data collection procedure as manual analysis is required to determine if valid firmware files are obtained from the web crawler. Lastly, the tool used to unpack the firmware images was successful across approximately 82% of the dataset, thereby leaving 18% of files unpacked and, subsequently not analyzed [8].

5.5.4 Technique: Differentially-Private Network Trace Analysis

Description of Technique
Several approaches have been developed to enable organizations to share data without privacy concerns. The most common way of sharing data between different organizations is to sanitize and anonymize the data. However, sanitization and anonymization may result in a dataset that is not useful because it is missing important information. Also, it has been shown that data anonymization is vulnerable to attacks. To overcome these privacy issues, differential privacy [13] was developed. Informally, the goal of differential privacy is to ensure that the presence or absence of records is not inferable from the output of an analysis. One of the advantages of differential privacy is that the strength, ϵ, of the privacy can be specified. As ϵ decreases, differential privacy becomes stronger.

The authors propose to apply differential privacy to network trace analysis. To do this, the authors base their technique [25] on the Privacy Integrated Queries (PINQ) platform. PINQ is a privacy-analysis platform that, in lieu of providing direct access to the underlying data, provides an abstraction over the data. This abstraction supports SQL-like queries, which an analyst specifies in a declarative language. PINQ supports two main types of operations: aggregations and transformations. Aggregation operations return aggregate values such as *Count* or *Sum*. Transformation operations return results that can be further analyzed, for example, results returned by a *Join* operation.

To test the efficacy of their approach, the authors test their approach with three datasets: Hotspot, IspTraffic, and IPscatter. The authors also use three privacy strength (ϵ) levels: 0.1, 1.0, and 10.0. The authors perform three analyses on the datasets: packet-level, flow-level, and graph-level. Under packet-level analyses, the authors found that their approach was effective in measuring distributions of packet sizes and ports. For example, for an ϵ-level of 0.1, the root mean squared error (RMSE) for packet size and ports is 0.01% and 0.07%, respectively. The authors' approach performed poorly for a more complex analysis such as worm fingerprinting, which is also a packet-level analysis. For example, for a total of 29 payloads, the authors' approach missed 74%, 17%, and 0% at ϵ-levels of 0.1, 01.0, and 10.0, respectively. Under flow-level analyses, the authors' technique had a false-positive rate of 90% at ϵ-level of 0.1 for detecting "stepping stones." Also, under graph-level analyses, the authors' technique had an RMSE of 0.17% and 50% at ϵ-level of 0.1 for anomaly detection and passive network discovery, respectively. Taken together, it can be inferred that as the privacy level becomes stronger (ϵ), the effectiveness of the approach decreases.

Technique-Specific Gaps
Although the results shown by the authors are promising, there are still significant gaps. One of the gaps is whether accurate complex analyses can be performed at higher privacy levels (ϵ). The authors acknowledge that it is not straightforward to implement some of the differentially private algorithms. The differentially private algorithms work on at-rest data. It is not clear whether differentially private algorithms can be built that work on network data in real time. It is also not clear how efficient the algorithms are in processing large amounts of network data.

5.5.5 Technique: Policy-Enhanced Private Set Intersection

Description of Technique
Private Set Intersection (PSI) is a cryptographic protocol that allows two parties, each having a set of elements (e.g., database records) to determine what subset they have in common without revealing those not in the subset. Policy-Enhanced Private Set Intersection (PPSI) [31] extends this functionality with support for policies over each element that must also be satisfied for that element to be used in the computation. These can be simple symmetric authorization policies, in which the the two parties comparing elements must have authorization from the element owner (e.g., patient consent for sharing a medical record). PPSI also supports more expressive policies, however. Asymmetric policies may have differing authorization requirements on either side (e.g. the IT head of each organization must authorize sharing of a record). Elements may be extended with attributes (such as classification rating) and policies can be defined over these governing when those elements can be shared. Elements may also be grouped into bundles, such that the bundle is listed in the intersection only if all of its elements are shared by both parties. Finally, disjunctions of authorizations are supported. These enable policies that are satisfied if any one of several authorizations is present.

PPSI has an additional nm computational overhead beyond the complexity of PSI itself, where n is the maximum number of elements per user and m is the maximum number of authorities per element. In empirical evaluation, this additional overhead added approximately one millisecond per additional authority, with a maximum set size of 2000 elements.

Technique-Specific Gaps
PPSI may not scale well to large datasets or numbers of authorities when more expressive policies are heavily used. No empirical results for extended policies are presented, but theoretical results increase complexity by one or more extra terms (e.g., $O(nmk)$ overhead).

5.5.6 Technique: Provable De-Anonymization of Large Datasets with Sparse Dimensions

Description of Technique
This technique provides an algorithm for de-anonymizing records in a large dataset, given noisy auxiliary information from another dataset [11]. Specifically, the paper considers an anonymized database where each record was arbitrarily perturbed, but no row or column was actually removed from the original dataset. Perturbed auxiliary information about a target record (the record to be de-anonymized) is also assumed to be available. The adversary's goal is to identify which of the anonymized records corresponds to the target record. The technique uses a weighted scoring algorithm to quantify the similarity of each record with shared attributes in the auxiliary data.

The authors mathematically prove bounds on two kinds of attack: Isolation and Information Amplification attacks. *Isolation Attacks* uniquely identify the target record. These are possible when the highest-scoring record in the anonymized dataset differs from the second highest by a certain threshold, the value of which is determined by how noisy the data is and how unique the target record is compared to others. *Information Amplification Attacks* do not uniquely identify the

target record, but allow the adversary to obtain more information about the target. This attack relies on the assumption that if multiple people agree on some rare attributes, they are likely to agree on others. This is similar to the assumptions made by recommendation systems like those used by Netflix and Amazon. By finding records that agree with known attributes of the target (taken from the auxiliary data), the technique can infer other attributes of the target record that are present in the anonymized data. Mathematical bounds on the effectiveness of the attack and quality of the learned information are provided, and depend on the degree of similarity between the target record and those used to "recommend" values of unknown attributes.

The authors empirically evaluate their attacks on an anonymized Netflix database. They find that for all but very high levels of perturbation, the large majority (60%–90%) of records are vulnerable to both isolation and information amplification attacks. Note that the attacks are not actually carried out. Instead, the authors verify that the records are sufficiently distinct from one another that de-anonymization is guaranteed to be possible given their theoretical bounds.

Technique-Specific Gaps

The technique relies on similarity metrics between attributes. This is not a problem for numeric data (e.g., ratings), but may be challenging for categorical data like IP addresses, hostnames, and port numbers. Another technique addressed in this paper (Measuring the Similarity of Network Hosts) attempts to mitigate this problem.

This page intentionally left blank.

6. SUMMARY OF RESEARCH DIRECTIONS AND PRIORITIZATION

A gap summary is provided in Table 1. Based on evaluating the state of the art in NKS systems and the priority of the gaps, we recommend the following research directions in the short-term. These directions focus on the highest priority gaps, which have a high likelihood of occurrence in modern NKS systems, and significantly impede our existing capabilities in this area (denoted by "H" in both the "Likelihood" and "Impact" columns in Table 1).

In the area of sensor placement, research should focus on sensors that collect a small, but indicative sample of network data. Since collection of all relevant data imposes high overhead that impedes effective collection and storage of such data, it is imperative that research is performed on data that provides symptomatic information about the network status and risk. Moreover, research should focus on connecting and binding identities across multiple layers for a more informed and actionable NKS. This should include, among other things, network identity (e.g., IP/port address), host identity (e.g., host name), application name, user identity (e.g., username), and user role.

In the area of data collection, short-term research should focus on minimizing the amount of data collected and on distributed systems for collection and storage of data. This can mitigate the existing trade-off between collecting meaningful data and the overhead imposed. In addition, research should focus on ensuring the authenticity of collected data or collecting indicative data that cannot easily be poisoned or tampered in a malicious environment. This possibility is ignored in many NKS systems, thus they can be ineffective in a malicious environment. Finally, the ideal semantic layer for each type of NKS data should be identified such that collection occurs within that layer. For example, identifying malware signatures is an uphill battle with respect to network packets because of the semantic gap (i.e., packets have to be reassembled and decoded). As a result, data collection about malware signatures should happen at the host layer, not the network layer. Although the ideal semantic layer is easy to reason about in many cases, it is ignored in numerous NKS systems.

In the area of data filtering, although high priority gaps do not exist, research can improve the efficiency of filtering by developing distributed, low resource filters. It is important to note that data filters compete for and consume the same resources (e.g., CPU cycles and memory) that are also used for data collection and, in many cases, analysis. As a result, any new research into data filters should be cognizant of this trade-off and attempt to develop decentralized, scalable filters.

Perhaps the most important component of an NKS system is the data analysis and sense-making component. Research in this area should focus on multiple directions. First, new algorithms that can resist evasion and can operate in a malicious environment with poisoned data should be designed and built. Second, data analysis and sense making algorithms must be tested in real-world networks with background traffic and activities in order to assess their real effectiveness. Evaluations performed in a closed, research-like environments should be viewed with proper skepticism. Third, research should focus on the development of fast data analytics that can be performed at line speed or those that have short time lags. Moreover, such analytics should be tweaked and adjusted based on the target network to ensure that the speed of natural changes in the enterprise network is much lower than the speed of data analysis and sense making.

Finally, in the area of information sharing, research should focus on proper data anonymization for specific NKS applications. Although, data anonymization is a difficult problem to solve, for specific NKS goals (e.g., sharing information about possible exfiltration activities), targeted algorithms can be designed that can anonymize sensitive data while preserving enough important information. Lastly, it is important to research systems that can share information about the nature of threat, rather than its artifacts and side effects. Since these artifacts can be different depending on how the attack manifests itself in a specific environment, sharing them can can be misleading to the peer organization. Thus, it is important to share "knowledge" or "intelligence" rather than raw data.

We believe that these research directions provide important short-term steps for improving NKS capabilities, network situational awareness, and information sharing.

TABLE 1

Table of NKS Components' Gap Prioritization

Component		Gap	Likelihood	Impact
Sensor Placement	1	Strong trade-off between how much data is collected and overhead	H	H
	2	Sensors require binding identities across multiple layers	H	H
	3	Sensors are spatially/temporally static	H	M
	4	Many network devices are black boxes	M	L
	5	Sensors require collection across multiple layers	M	L
Data Collection	1	Collection at line speed is very hard	H	H
	2	Assumes a benign environment	H	H
	3	Collection occurs at the wrong semantic level	H	H
	4	Analysis does not happen at collection location, resulting in transfer overhead	H	M
	5	Unscalable to store collected data	H	M
	6	Efficient techniques depend on type of data collected	M	M
	7	Data must be collected to find interesting events, but one must know what data to collect to find events — a Catch-22	M	M
	8	In-band collection perturbs the network	H	L
	9	Requires manual effort to specify what data to collect	M	L
	10	Cannot work on encoded/compressed data	M	L
Data Filtering	1	Trade-off between resources and quality of data filtering	H	M
	2	Filtering relies on thresholds from ground-truth, which may be unknown	H	M
	3	Advanced filtering techniques require customized infrastructure	M	H
	4	Efficient filters are shallow, while interesting features require deep inspection	M	M
	5	Filtering does not necessarily reduce storage size	M	L
Data Analysis and Sense-Making	1	Requires thresholds for anomaly detection, which are manual, can be evaded, and need ground truth	H	H
	2	Often evaluated in a closed world with no noise	H	H
	3	Network conditions change faster than analysis convergence	H	H
	4	Complex analytics are not scalable	H	M
	5	Window-based analysis is limited as large windows are unscalable, but small windows are easy to evade	M	M
	6	Often requires out-of-band information	M	M
	7	Small packets and fragmentation degrade performance	M	M
	8	Relies on higher-level identities, e.g., user ID instead of IP address	M	M
	9	Assumes a benign environment and is vulnerable to poisoning	L	H
	10	Often assumes the majority of devices are healthy	L	M

	11	More effective techniques are active, visible to attackers, and perturb the network	L	L
Information Sharing	1	Anonymizing data is hard as structure must be preserved and identifying information removed	H	H
	2	Attack campaigns have different manifestations in different environments	H	H
	3	Open-source data is over-represented due to availability	H	M
	4	Real-time sharing is very resource-intensive	M	H
	5	Biased towards security-conscious organizations	M	M

7. CONCLUSION

In this report, we have identified the major gaps in building a network knowledge synthesis (NKS) system. The vision of an NKS system is to achieve effective situation awareness of the network for improved network defense and timely information sharing with peer organizations. Such a system will include five major components: Sensor Placement, Data Collection, Data Filtering, Data Analysis and Sense-Making, and Information Sharing. We identified the high priority gaps in building an NKS system by reviewing the state-of-the-art techniques proposed or built in this domain, extracting their implicit or explicit gaps and weaknesses, and ranking those gaps based on their likelihood and impact in modern enterprise networks. We believe that by focusing the short-term research on the high priority gaps, the community can make major headway in achieving the vision of effective situational awareness and information sharing.

This page intentionally left blank.

REFERENCES

[1] Bernhard Amann, Robin Sommer, Aashish Sharma, and Seth Hall. A lone wolf no more: supporting network intrusion detection with real-time intelligence. In *Research in Attacks, Intrusions, and Defenses*, pages 314–333. Springer, 2012.

[2] Johanna Amann, Seth Hall, and Robin. Count me in: Viable distributed summary statistics for securing high-speed networks. In *In Proceedings of 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 320–340, September 2014.

[3] Manos Antonakakis, David Dagon, Xiapu Luo, Roberto Perdisci, Wenke Lee, and Justin Bellmor. A centralized monitoring infrastructure for improving dns security. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, RAID'10, pages 18–37, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] George Baah, Thomas Hobson, Hamed Okhravi, Shannon Roberts, William Streilein, and Sophia Yuditskaya. Gaps in cyber defense automation. 2015.

[5] Kevin Bauer, Thomas Hobson, Hamed Okhravi, Shannon Roberts, and William Streilein. Gaps in defensive countermeasures for web security. 2015.

[6] Antonio Bianchi, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Blacksheep: Detecting compromised hosts in homogeneous crowds. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 341–352, New York, NY, USA, 2012. ACM.

[7] Jie Chu, Zihui Ge, Richard Huber, Ping Ji, Jennifer Yates, and Yung-Chao Yu. ALERT-ID: Analyze Logs of the Network Element in Real Time for Intrusion Detection. In *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses*, RAID'12, pages 294–313, Berlin, Heidelberg, 2012. Springer-Verlag.

[8] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 95–110, San Diego, CA, 2014. USENIX Association.

[9] Scott E Coull, Fabian Monrose, and Michael Bailey. On measuring the similarity of network hosts: Pitfalls, new metrics, and empirical analyses. In *NDSS*, 2011.

[10] Ang Cui, Jatin Kataria, and Salvatore J. Stofo. From Prey to Hunter: Transforming Legacy Embedded Devices into Exploitation Sensor Grids. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 393–402, New York, NY, USA, 2011. ACM.

[11] Anupam Datta, Divya Sharma, and Arunesh Sinha. Provable de-anonymization of large datasets with sparse dimensions. In *Principles of Security and Trust*, pages 229–248. Springer, 2012.

[12] Sean Donovan and Nick Feamster. Intentional network monitoring: Finding the needle without capturing the haystack. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, HotNets-XIII, pages 5:1–5:7, New York, NY, USA, 2014. ACM.

[13] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, volume 4052, pages 1–12, July 2006.

[14] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 332–346. IEEE, 2012.

[15] Open Networking Foundation. Openflow switch specification v1. 3.0, 2012.

[16] Paul Giura and Nasir Memon. Netstore: An efficient storage infrastructure for network forensics and monitoring. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, RAID'10, pages 277–296, Berlin, Heidelberg, 2010. Springer-Verlag.

[17] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *Proc. NSDI*, 2014.

[18] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The parrot is dead: Observing unobservable network communications. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 65–79. IEEE, 2013.

[19] Heqing Huang, Su Zhang, Xinming Ou, Atul Prakash, and Karem Sakallah. Distilling critical attack graph surface iteratively through minimum-cost sat solving. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

[20] Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. Kargus: A highly-scalable software-based intrusion detection system. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 317–328, New York, NY, USA, 2012. ACM.

[21] Mobin Javed and Vern Paxson. Detecting stealthy, distributed ssh brute-forcing. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 85–96, New York, NY, USA, 2013. ACM.

[22] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 99–111, Lombard, IL, 2013. USENIX.

[23] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *NSDI*, pages 113–126, 2012.

[24] Srinivas Krishnan, Kevin Z. Snow, and Fabian Monrose. Trail of bytes: Efficient support for forensic analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 50–60, New York, NY, USA, 2010. ACM.

[25] Frank McSherry and Ratul Mahajan. Differentially-private network trace analysis. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 123–134, New York, NY, USA, 2010. ACM.

[26] Masoud Moshref, Minlan Yu, and Ramesh Govindan. Resource/accuracy tradeoffs in software-defined measurement. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 73–78. ACM, 2013.

[27] Hamed Okhravi, Thomas Hobson, Chad Meiners, and William Streilein. A study of gaps in attack analysis. 2014.

[28] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Comput. Netw.*, 31(23-24):2435–2463, December 1999.

[29] Phillip Porras, Steven Cheung, Martin Fong, Keith Skinner, and Vinod Yegneswaran. Securing the software-defined network control layer. In *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS), San Diego, California*, 2015.

[30] Seungwon Shin, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.

[31] Emil Stefanov, Elaine Shi, and Dawn Song. Policy-enhanced private set intersection: Sharing information while enforcing privacy policies. In *Public Key Cryptography–PKC 2012*, pages 413–430. Springer, 2012.

[32] Nedim Šrndic and Pavel Laskov. Practical evasion of a learning-based classifier: A case study. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 197–211, Washington, DC, USA, 2014. IEEE Computer Society.

[33] Andreas Wundsam, Dan Levin, Srini Seetharaman, Anja Feldmann, et al. Ofrewind: Enabling record and replay troubleshooting for networks. In *USENIX Annual Technical Conference*, 2011.

[34] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 199–208, New York, NY, USA, 2013. ACM.

[35] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *NDSS*, 2012.

[36] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[37] Ye Yu, Chen Qian, and Xin Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 85–90. ACM, 2014.

This page intentionally left blank.

This page intentionally left blank.